AD/ADVANTAGE

Component Management Facility

P19-2131-01

CINCOM®
The Smart Choice®

AD/Advantage® Component Management Facility

Publication Number P19-2131-01

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, OH 45246-3732
U. S. A.

PHONE: (513) 612-2300
FAX: (513) 612-2000
WORLD WIDE WEB: http://www.cincom.com

Attention:

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

Release information for this manual

The *AD/Advantage Component Management Facility*, P19-2131-01, is dated November 30, 1998. This document supports Release 1.1 of AD/Advantage, which works with Release 2.4 of MANTIS for the VMS and UNIX environments and with Release 1.1 of Ad/Advantage.

We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. A Reader Comment Sheet is included at the end of the manual for your convenience.

Cincom Technical Support for AD/Advantage

FAX: (513) 612-2000
Attn: MANTIS Support

E-mail: helpna@cincom.com

Phone: 1-800-727-3525

Mail: Cincom Systems, Inc.
Attn: MANTIS Support
55 Merchant Street
Cincinnati, OH 45246-3732
U. S. A.

Contents

About this book

Using this document

MANTIS is an application development system that consists of design facilities (e.g., screens and files) and a programming language. describes the Component Management Facility of AD/Advantage, which enables you to create MANTIS programs that include reusable subroutines.

Document organization

The information in this manual is organized as follows:

Chapter 1—Accessing the Component Management Facility (CMF)
Describes how to access the component Management Facility in AD/Advantage.

Chapter 2—Designing and modifying components
Describes components and how to modify them.

Chapter 3—Source programs
Describes source programs and how to create and modify them.

Chapter 4—Composing programs
Describes the Compose process.

Chapter 5—Decomposing programs
Describes the Decompose process.

Index

Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example

`Constant width type`	Represents screen images and segments of code.	`Screen Design Facility`
		`GET NAME LAST`
		`INSERT ADDRESS`

Slashed b (ⱥ)	Indicates a space (blank). The example indicates that a password can have a trailing blank.	`WRITEPASSⱥ`
Brackets [ ]	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations.	
	A single item enclosed by brackets indicates that the item is optional and can be omitted. The example indicates that you can optionally enter a program name.	`COMPOSE [`*`program-name`*`]`
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected. The example indicates that you can optionally enter NEXT, PRIOR, FIRST, or LAST. (NEXT is underlined to indicate that it is the default.)	**NEXT** **PRIOR** **FIRST** **LAST**

Convention	Description	Example
Braces { }	Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select. The example indicates that you must enter FIRST, LAST, or a value for *begin*.	**FIRST** ***begin*** **LAST**
Underlining (In syntax)	Indicates the default value supplied when you omit a parameter. The example indicates that if you do not specify ON, OFF, or a row and column destination, the system defaults to ON.	**SCROLL** [**ON** **OFF** [***row***] [**,*col***]]
	Underlining also indicates an allowable abbreviation or the shortest truncation allowed. The example indicates that you can enter either PRO or PROTECTED.	PROTECTED
Ellipsis points...	Indicate that the preceding item can be repeated. The example indicates that you can enter (A), (A,B), (A,B,C), or some other argument in the same pattern.	(*argument*,...)

Convention	Description	Example
UPPERCASE	Indicates MANTIS reserved words. You must enter them exactly as they appear. The example indicates that you must enter CONVERSE exactly as it appears.	`CONVERSE name`
Italics	Indicate variables you replace with a value, a column name, a file name, and so on. The example indicates that you can supply a name for the program.	`COMPOSE [program-name]`
Punctuation marks	Indicate required syntax that you must code exactly as presented. ( ) parentheses . period , comma : colon ' ' single quotation marks	[LET] *v* [(*i*) / (*i*, *j*)] [ROUNDED(*n*)] = *e1* [, *e2*, *e3*...]

MANTIS documentation series

MANTIS is an application development system designed to increase productivity in all areas of application development, from initial design through production and maintenance. MANTIS is part of AD/Advantage, which offers additional tools for application development. Listed below are the manuals offered with MANTIS in the IBM mainframe environment, organized by task. You may not have all the manuals listed here.

Getting started

- *MANTIS Startup and Configuration Guide (MVS/CICS)*, P19-5018
- *MANTIS Startup and Configuration Guide (VSE/CICS)*, P19-5019

General use

- *MANTIS Language*, P19-5002
- *MANTIS Quick Reference*, P19-5003
- *MANTIS Master Glossary*, P19-5009
- *MANTIS Object Design, Prototyping, and Print Facilities*, P19-5001
- *MANTIS Messages and Codes*, P19-5004*
- *MANTIS SQL Support for DB2 and SQL/DS—User Supplement*, P19-3107
- *MANTIS Program Design and Editing*, P19-5013
- *MANTIS for DL/1—User Supplement*, P19-5008
- *MANTIS SQL Support for SUPRA IBM User Supplement*, P19-3105
- *IMS MANTIS—User Supplement*, P19-3101

- *AD/Advantage User's Guide*, P19-7001
- *AD/Advantage Messages*, P19-7003*
- *SAP Component User's Guide*, P19-7000
- *XREF Online User's Guide*, P19-0011
- *XREF Administration*, P19-0012
- *Entity Transformers*, P19-0013
- *CASE Integration Facility*, P19-0020
- *Design Object Generators From Catalogs*, P19-0021

Master user tasks

- *MANTIS Administration Guide*, P19-5005
- *AD/Advantage Administration*, P19-7002
- *MANTIS Messages and Codes*, P19-5004*
- *AD/Advantage Messages*, P19-7003*

Manuals marked with an asterisk (*) are listed twice because you use them for different tasks.

Educational material

AD/Advantage and MANTIS educational material is available from your regional Cincom education department.

1

Accessing the Component Management Facility (CMF)

The AD/Advantage Component Management Facility (CMF) incorporates the software methodology of Component Management, allowing you to create MANTIS programs that include reusable subroutines called components. Components are subroutines that perform a specific function common to more than one program. Components are the "building blocks" of your application because they can be used and reused as necessary at various locations. Components usually have value in more than one application design, but are often difficult to locate and use in complex applications.

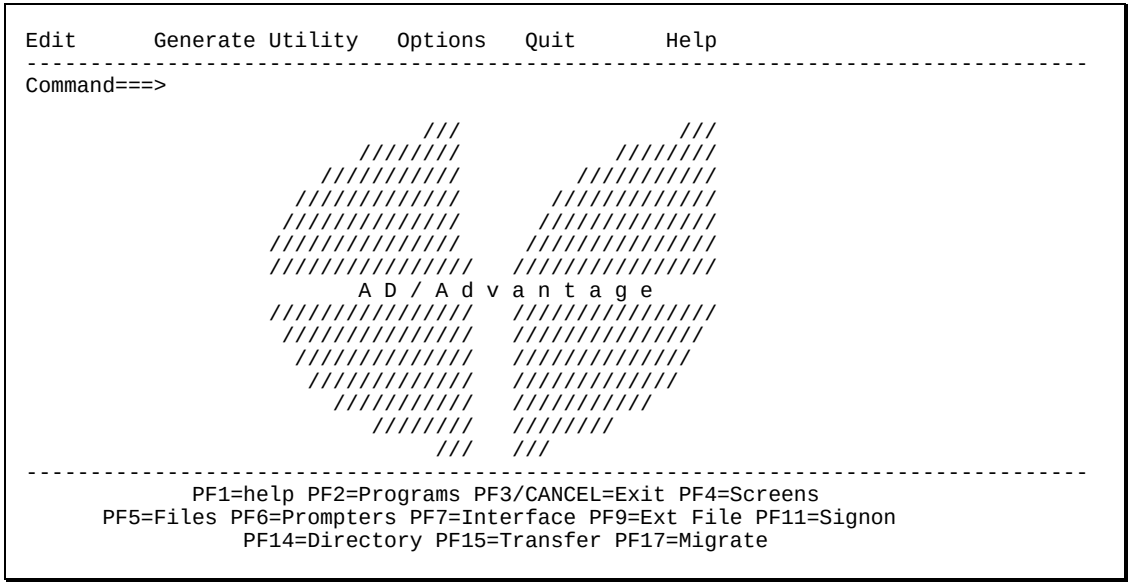
Programs created using CMF are known as component-engineered programs. You can create new component-engineered programs, or you can modify existing programs to contain component-engineered code (thus making them component-engineered programs).

This manual contains information on:

- Components in "Designing and modifying components" on page 19
- Source programs in "Source programs" on page 27
- Composing programs in "Composing programs" on page 31
- Decomposing programs in "Decomposing programs" on page 35

Facility Selection Menu

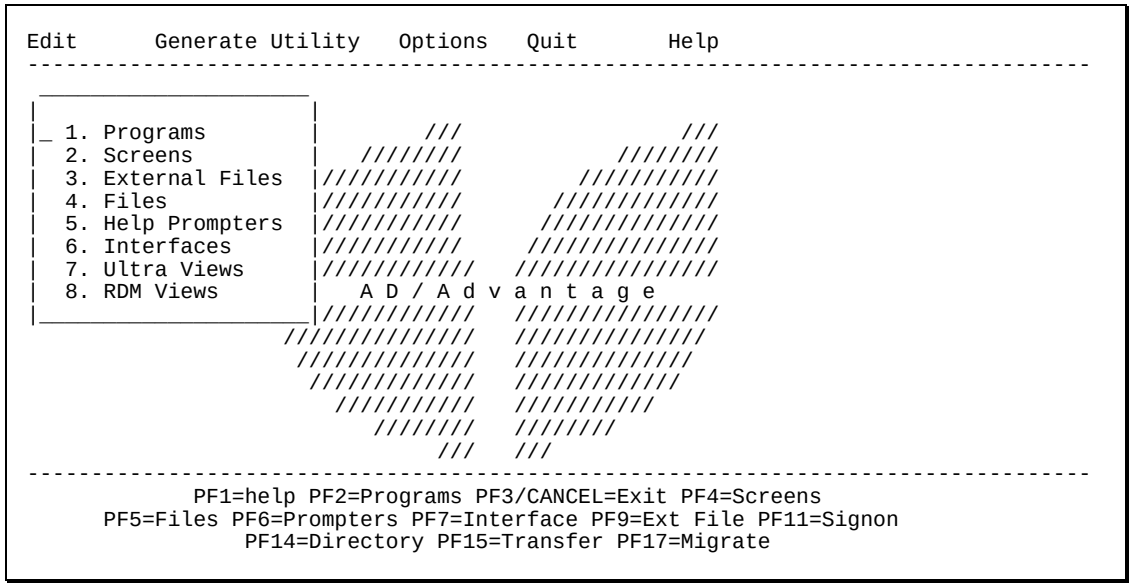
All AD/Advantage facilities are provided as options on the AD/Advantage Facility Selection menu, shown below:



The AD/Advantage Facility Selection Menu on your system may be different from the preceding example if your Master User has customized it for your installation. A Master User has access to certain facilities and information not available to all MANTIS users.

Entity Transformers

To access CMF, select Entity Transformers by entering “E” in the command field. The Edit pull-down menu shown below will appear.

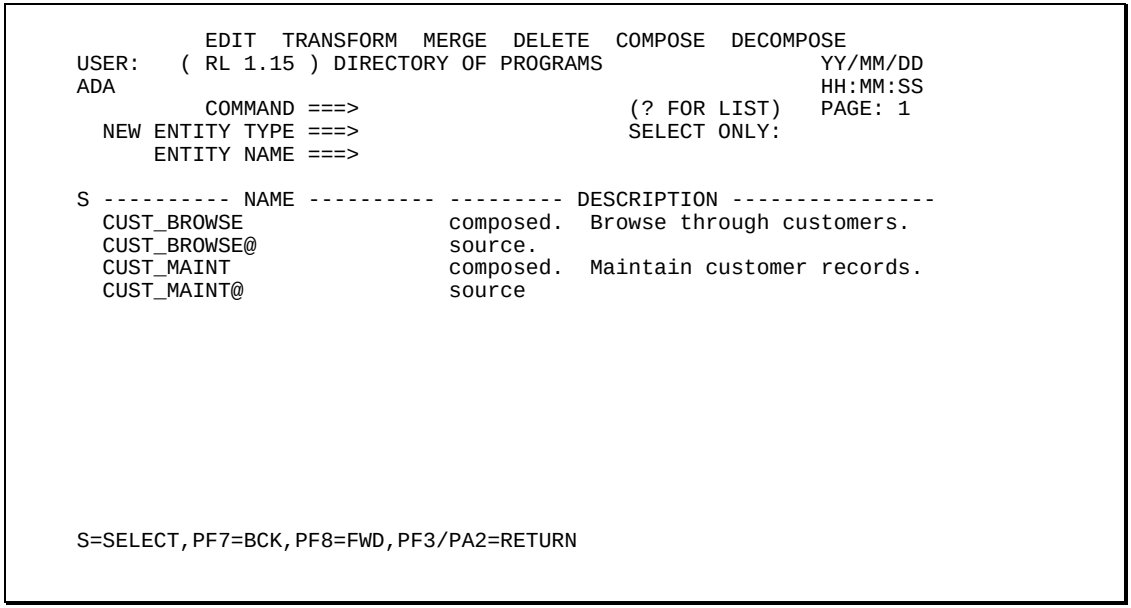


The edit pull-down menu on your system depends on your environment. For complete information on Entity Transformers, refer to *Entity Transformers User Supplement*, P19-0013.

Select programs from this screen by entering P or 1.

Directory of Programs

The Directory of Programs displays with a list of programs and descriptions at the bottom of the screen. The available functions are displayed across the top of the directory screen. Enter the CMF functions (COMPOSE and DECOMPOSE) in the COMMAND field at the top of the screen.



COMMAND

Description *Optional*. Specifies the name of the command you want to perform.

Options COMPOSE

DECOMPOSE

Considerations

- You can use the Compose and Decompose actions on programs only. This field is also used for performing functions on other entities through Entity Transformers, which is explained in *Entity Transformers User Supplement*, P19-0013.
- For more information on the Compose and Decompose process, see "Composing programs" on page 31 and "Decomposing programs" on page 35.

NEW ENTITY TYPE

Description *Optional*. Specifies the type of entity to receive the action.

Consideration You can use the Compose and Decompose actions on programs only. This field is also used for performing functions on other entities through Entity Transformers, which is explained in *Entity Transformers User Supplement*, P19-0013.

ENTITY NAME

Description *Optional*. Specifies the name of the entity receiving the action.

Consideration For COMPOSE and DECOMPOSE, this field accepts program names only.

S

Description *Optional*. Specifies which programs you want to Compose or Decompose. You enter an S (for select) next to the program name to Compose or Decompose the program.

The next chapter explains how to build and use components.

2

Designing and modifying components

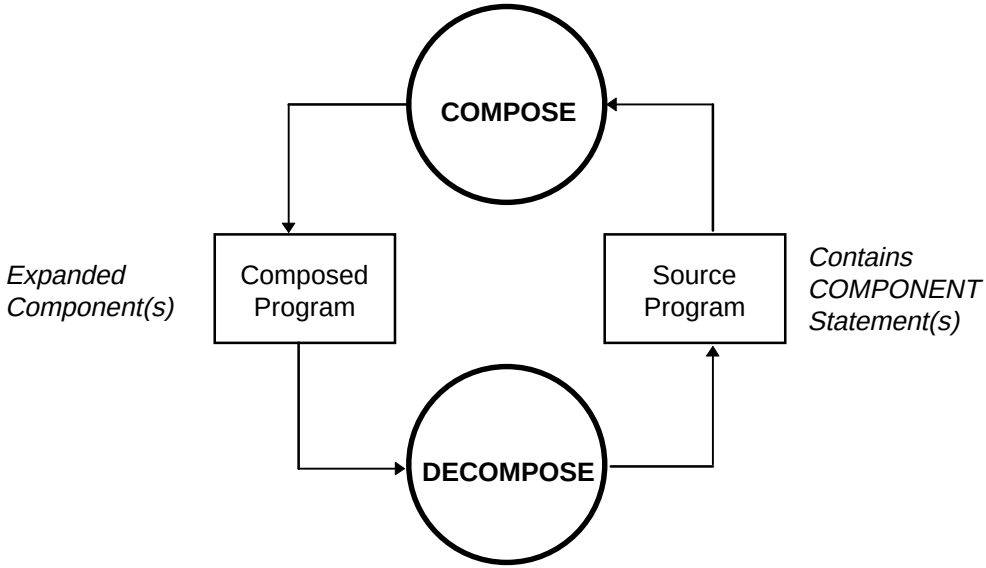
Components are subroutines that perform a specific function common to more than one program. Components, specified by the COMPONENT statement, simplify your coding effort by allowing the optimal use, reuse, and management of programs. You can modify your components individually or directly in a composed program.

A source program contains MANTIS source code and at least one COMPONENT statement. You cannot execute source programs. CMF provides the Compose process, which assembles (composes) a source program containing COMPONENT statements and related component code into a composed program that you can edit and run.

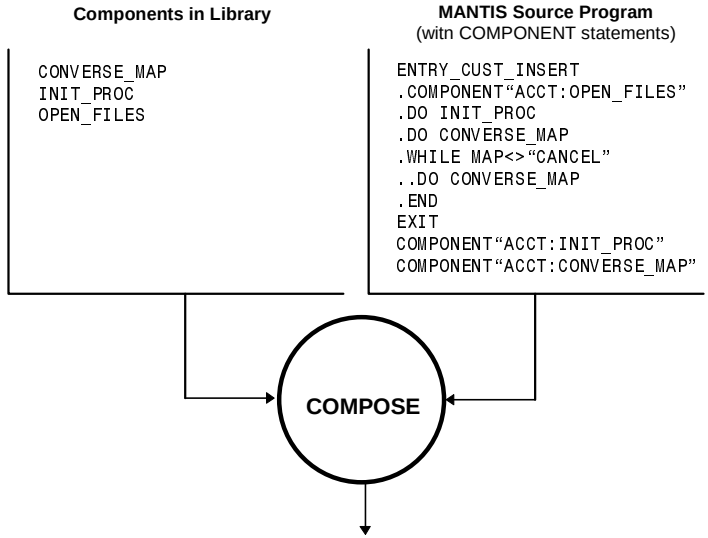
A composed program is an executable program that is the result of issuing the COMPOSE command on a source program. An executable program is any program that can be run in MANTIS. As a result, all composed programs are executable programs, but not all executable programs are composed.

You can make changes directly to the composed program and issue the CMF DECOMPOSE command. The Decompose process reverses the Compose process and disassembles the composed program into source code and component code.

The following figure provides an overview of the relationship between source programs, the Compose process, composed programs, and the Decompose process.



The following figure shows how CMF works. Three components originate in the library: CONVERSE_MAP, INIT_PROC, and OPEN_FILES. The components are coded into a MANTIS source program using three COMPONENT statements. The COMPOSE command is issued on the source program, resulting in a composed program that you can edit and run.



Composed (Executable) MANTIS Program

```
ENTRY_CUST_INSERT
|*COMPONENT“ACCT:OPEN_FILES”
.FILE RECORD(“ACCT:CUST_MAINT”)
.SCREEN MAP (“CUST_MAINT”)
|*CEND
.DO INIT_PROC
.DO CONVERSE_MAP
.WHILE MAP<>“CANCEL”
..DO CONVERSE_MAP
.END
EXIT
|*COMPONENT“ACCT:INIT_PROC”
ENTRY INIT_PROC
.
.
EXIT
|*CEND
|*COMPONENT“ACCT:CONVERSE_MAP”
.
.
|*CEND
```

To change one of the components in the preceding illustration, make the change directly in the composed program and then issue the DECOMPOSE command. The DECOMPOSE command reverses the Compose process and splits the composed program into source code and component code.

Designing components

The first step in designing and coding components is determining which subroutines in your application would make good components. The components you select may already exist as subroutines in your application, or you may want to code new ones. Once you decide which subroutines will become components in your application, you code COMPONENT statements in a source program. The COMPONENT statement indicates that the component code will be pulled into the program when you issue the COMPOSE command on the source program.

A well-designed component will be used throughout your application in more than one program. A good example of a component is the standard subroutine used for the check-digit computation of account numbers in the banking industry. This subroutine is a typical component because it appears in more than one program of a banking application. A date conversion routine and an initialization routine are two more examples of components.

You can use existing subroutines as components, or you can code new ones. Like any other MANTIS program, a component is stored in your library. You can store components in a common library or in separate libraries.

If you store components in a common library, it is important to establish naming standards to distinguish between programs and components. This document identifies components with the prefix “CUS”, for example, CUS_INIT_HEADER.

Once you create components, they reside in a program library. You use them by coding a COMPONENT statement in a source program for each component you want to use. Code the COMPONENT statements in the order in which the components are to be brought into the program. (For information on coding the COMPONENT statement in a source program, see "Source programs" on page 27). When you compose the source program, the MANTIS source code and the component code are assembled into a separate, composed program that you can run and edit. For information on the Compose process, see "Composing programs" on page 31.

General considerations

- Components can reside in any library as long as the COMPONENT statement contains the correct password.
- COMPONENT statements in the source program must have a corresponding component in the library before you compose the source program, or an error message will be displayed.
- A COMPONENT statement cannot refer to a program that contains a COMPONENT statement, This means components are single-level only (one level between the components and the source programs). If a component contains a COMPONENT statement, that statement will be ignored when you compose the program. If you attempt to execute the composed program, the ignored COMPONENT statement causes the program to fail.

Modifying components

CMF provides two ways to modify component code:

- Editing the individual component code in the editor, replacing it, and composing the source program.
- Editing the component code directly in a composed program using the editor, replacing it, and issuing the DECOMPOSE command.

These two methods are discussed in detail below:

Individual component code. To modify individual component code, follow these steps:

1. Select the component from your library and use the editor to make changes. When you change individual components in the source program, the composed program no longer reflects the updated component code.
2. Issue the COMPOSE command on the source program to apply the component changes to the composed program. (See “Composing programs” on page 31 for information on the Compose process.)

Expanded component code. To change the expanded component code in a composed program, follow these steps:

1. Select the composed program from your library and use the editor to make changes directly to the expanded component code.

2. Nominate the components to be decomposed by coding the at sign (@) character in place of the asterisk (*) in the commented COMPONENT statement of the program. The vertical bar (|) must remain as the first character and the at sign (@) character must be the second character in the commented COMPONENT statement.

3. If you made changes to MANTIS source code and components (or source code only), you must supply a SOURCE statement nominated with the at sign (@) in the composed program before you decompose it. MANTIS will then create/replace the source program as well as any nominated components. See "Using SOURCE statements in executable programs" on page 39 for information on the SOURCE statement.

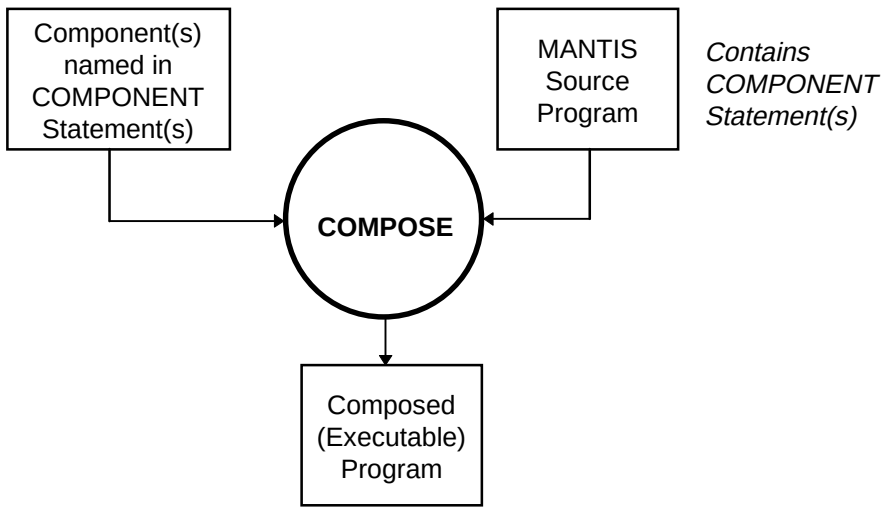
4. Issue the DECOMPOSE command on the composed program (DECOMPOSE is discussed in "Decomposing programs" on page 35). DECOMPOSE recognizes the "|@" as the component nominated to be decomposed, and the |*CEND statement as the end of the nominated component. No other components will be affected unless you mark them this way.

3

Source programs

Source programs contain MANTIS source code and at least one COMPONENT statement. You cannot execute source programs. You can create new source programs by using the editor. You can also modify existing executable programs to contain blocks of code bounded by |@COMPONENT/|*CEND statements and then issue the DECOMPOSE command to split the program into source code and component code. This method is discussed in “Decomposing programs” on page 35. The advantage of the first method is that its top-down design, which starts with creating the source program, provides an overall structure of which the components become a part.

Source programs are used as input with the COMPOSE command (described in “Composing programs” on page 31). The following figure shows the starting position of a source program in the Compose process:



Naming source programs

CMF uses the at sign (@) character as the system default value to identify source programs in your library. The at sign (@) is appended to the end of source program names.

When you issue the COMPOSE command to assemble the source program and components into a composed program, CMF will recognize the at sign (@) and automatically assign the composed program with the same name as the source program, minus the at sign (@). For example, if you have a source program named CUST_BROWSE@, the composed program will be named CUST_BROWSE.

The following figure shows the Program Directory List displaying source programs and composed programs.

```
          EDIT  TRANSFORM  MERGE  DELETE  COMPOSE  DECOMPOSE
USER:   ( RL 1.15 ) DIRECTORY OF PROGRAMS                YY/MM/DD
ADA                                                      HH:MM:SS
          COMMAND ===>                     (? FOR LIST)   PAGE: 1
  NEW ENTITY TYPE ===>                     SELECT ONLY:
      ENTITY NAME ===>

S ---------- NAME ---------- --------- DESCRIPTION ----------------
  CUST_BROWSE               composed.  Browse through customers.
  CUST_BROWSE@              source.
  CUST_MAINT                composed.  Maintain customer records.
  CUST_MAINT@               source




S=SELECT,PF7=BCK,PF8=FWD,PF3/PA2=RETURN
```

If you do not use the at sign (@) for source program names, use the terms “source,” “composed,” or “executable” in the description text for the program to distinguish between them.

Coding source programs

You can code a new source program using the editor. Your source program will contain MANTIS source code and at least one COMPONENT statement. The COMPONENT statement is required and identifies each component you want to use in the application.

The COMPONENT statement identifies each component you want to use in a MANTIS source program. The COMPONENT statement for source programs is coded as follows:

COMPONENT "[*library*:]*component-name*[/*password*][/*description*]"

You can code COMPONENT statements in most places in your MANTIS program, but not as the first statement if the source program contains ENTRY/EXIT statements. During the Compose process, the COMPONENT statement is commented (in the composed program) with the vertical bar (|). As the first statement of the program, a commented line makes the MANTIS program non-executable if it precedes an ENTRY statement.

For consistency, COMPONENT statements can be coded following the last EXIT statement at the end of your MANTIS source program (as shown in the examples in this manual) if the components consist of subroutines executed from the mainline code.

Component code will be placed in the composed program in the same order that the COMPONENT statements appear in the source program.

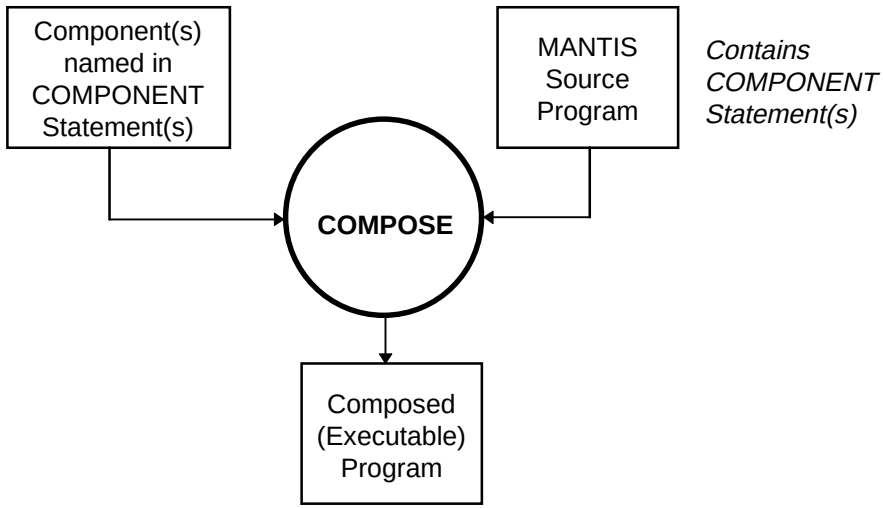
If the components consist of subroutines bounded by ENTRY/EXIT statements, never specify the COMPONENT statement within the ENTRY/EXIT statements of the mainline routine. The Compose process produces nested ENTRY/EXIT pairs, which is not allowed in MANTIS.

Notice that double quotes (") are required around the parameters of the COMPONENT statement, as shown in the examples. Also, note that COMPONENT statements cannot be continued on subsequent lines. (See "Directory of Programs" on page 16 for an example of a source program containing components.)

4

Composing programs

The Compose process assembles, or composes, a source program and component code into a composed MANTIS program that you can edit and run. The following figure illustrates the Compose process.



The COMPOSE command is issued on a source program (containing COMPONENT statements) only. The components for each COMPONENT statement must exist in your library at the time you issue the COMPOSE command, or you will receive an error message. The steps for creating a new source program and components for issuing the COMPOSE command are:

1. Code a new source program using the editor and include COMPONENT statements (at least one) to identify each component you want to use.

2. Save your changes in the editor and exit from the editing session.

3. Create or modify (if they already exist) each component specified in the source program by using the editor. Save your changes and exit from the editing session.

4. Start the COMPOSE on the source program.

   First, enter COMPOSE in the COMMAND field. Now select with an "S" a program containing COMPONENT statements and press RETURN.

   After you press RETURN, CMF issues a confirmation message containing the name of the chosen program, so you can confirm to continue or stop the Compose process. When you confirm the command, a composed (executable) program with its components is created.

Once the composed program is created, you can continue to make changes to the source program and issue the Compose command to replace a current composed program as necessary. You can repeat this cycle as often as needed when you alter MANTIS source programs or components.

General considerations

- You can issue COMPOSE on source programs only.
- There must be at least one COMPONENT statement in the source program for the Compose process to work.
- If the at sign (@) is appended to the source program name, such as CUST_BROWSE@, the composed program will have the same name as the source program without the at sign (@), for example, CUST_BROWSE.
- Currently only one component level is supported with the Compose process. This means that if a component contains a COMPONENT statement, the component will not resolve completely when you issue COMPOSE on the source program. Then, if you attempt to execute the composed program, the ignored COMPONENT statement could cause the program to fail.

5

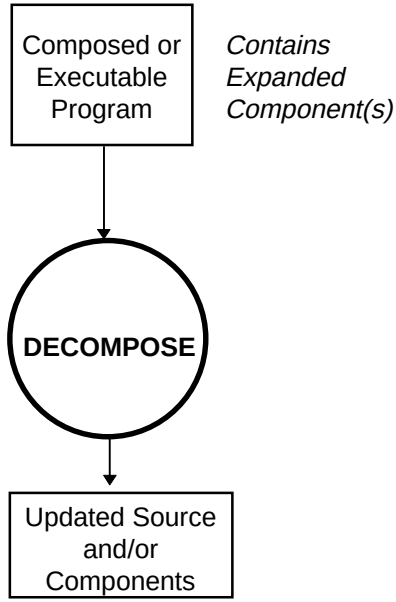
Decomposing programs

The Decompose process disassembles a composed program or an executable program, splitting MANTIS source code and component code and updating your library with any changes.

Use the Decompose process as follows:

- On a composed program (one created as a result of the Compose process) when you have made changes to either the component code or the source code.
- On an executable program (any program that can be run in MANTIS) as part of the process of modifying a program that already exists in your library into a component-engineered program.

The Decompose process starts with an executable or composed program as shown in the following figure:



To start the Decompose process:

1. Enter DECOMPOSE in the COMMAND field.

2. Select with an “S” a composed program in which components have been created or modified and need to be saved or replaced. Press RETURN, This will save/replace the components created/modified.

Nominating components to be decomposed

The first step in decomposing a composed program or an executable program is nominating, or marking, components (new and existing) in the program with the at sign (@). This enables the Decompose process to recognize the components to be decomposed.

To nominate components, code the at sign (@) in place of the asterisk (*) in new or modified commented COMPONENT statements of the executable or composed program. The vertical bar (|) must remain as the first character and the at sign (@) must be the second character in the statement. You do not have to change the |*CEND statement in any way for an existing component. However, a |*CEND statement is required if you create a new component.

The following code listing shows an example of a commented COMPONENT statement in line 550 nominated to be decomposed.

```
 10 ENTRY CUST_INSERT
          .
100 .DO INIT_PANEL_TITLE
          .
540 EXIT
550 |@COMPONENT “ACCT:CUS_INIT_HEADER”
560 ENTRY INIT_PANEL_TITLE
570 .MAP_TITLE=“INSERT A NEW CUSTOMER”
580 .MAP_DATE=DATE
590 .MAP_TIME=TIME
600 .MAP_FUNCTION=FUNCTION_TYPE
610 EXIT
620 |*CEND
```

You can nominate one or more existing components as shown. you can also create a new component, nominate it, and include a |*CEND statement. Decompose replaces existing components and saves new ones in your library. After the program is decomposed, the at sign (@) will be replaced by the asterisk (*) in the commented COMPONENT statement.

Decomposing executable programs

You can create a component-engineered program using an executable program currently in your library by using the Decompose process. Follow these steps:

1. Nominate the components in the executable program by coding |@COMPONENT statements and |*CEND statements around the code representing the desired component. These statements must be present in the executable program for the Decompose process to work. See “Nominating components to be decomposed” on page 37 for information on nominating components.

2. Be sure that the at sign (@) is in the second position of the commented COMPONENT statement. This at sign nominates the commented statement to be created (if new) or updated (if existing) in your library.

3. Code a SOURCE statement and nominate it with the at sign (@) so the Decompose process recognizes the MANTIS source code.

4. Save the changes made in the editor and exit from the editing session.

5. Issue the DECOMPOSE command on the executable program to split the source code and the component code.

Once the composed program is created, you can continue to make changes to the source program and issue the COMPOSE command to replace a current composed program as necessary. You can repeat this cycle as often as needed when you alter MANTIS source programs or components.

Using SOURCE statements in executable programs

Code the SOURCE statement in an executable program to name the library (your library only), program, password, and description of the source program to be created or replaced by the Decompose process. The SOURCE statement is coded as follows:

|@SOURCE "[*library*:]*program-name*[/*password*][/*description*]"

The SOURCE statement must be nominated to be recognized by the Decompose process. Nominate a SOURCE statement by coding the at sign (@) in the second position following the vertical bar (|), for example, |@SOURCE. In addition, be sure you code double quotes (") around the parameters of the SOURCE statement.

The following table shows the conditions when the SOURCE statement is required.

MANTIS source changes?	Component changes	|@SOURCE statement required?
Yes	No	Yes
No	Yes	No
No	No	No
Yes	Yes	Yes

If you want to issue a Decompose and do not want SOURCE replaced, you can replace the at sign (@) with an asterisk (*) so that the source statement does not need to be removed from the program.

Decomposing composed programs

You can use the Decompose action on a composed program when you have made changes in the program either to source code or component code.

To decompose a composed program, follow these steps:

1. Use the editor to modify a composed program.

2. Nominate the components that you modified by changing the asterisk (*) to the at sign (@) on affected COMPONENT statements. (See “Nominating components to be decomposed” on page 37 for information on nominating components.)

   This is all you have to do if you changed component code. You do not have to alter |*CEND statements. However, if you changed source code, be sure that you include a nominated SOURCE statement.

3. Save the changes made in the editor and exit from the editing session.

4. Start the DECOMPOSE command on the composed program. First, enter DECOMPOSE in the COMMAND field. Now select with an “S” the composed program and press RETURN.

After you press RETURN, CMF issues a confirmation message containing the name of the composed program, so you can confirm to continue or stop the Decompose process.

Index

P

S

Reader Comment Sheet

Name: _____

Job title/function: _____

Company name: _____

Address: _____

Telephone number: _____ Date: _____

How often do you use this manual? ☐ Daily ☐ Weekly ☐ Monthly ☐ Less

How long have you been using this product? ☐ Months ☐ Years

Can you find the information you need? ☐ Yes ☐ No Please comment.

_____

_____

Is the information easy to understand? ☐ Yes ☐ No Please comment.

_____

_____

Is the information adequate to perform your task? ☐ Yes ☐ No Please comment.

_____

_____

General comment: _____

_____

_____

_____

_____

_____

To respond, please fax to Larry Fasse at (513) 612-2000.